import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import EarlyStopping

# Function to load and preprocess data

def load\_and\_preprocess\_data(data\_p/content/drive/MyDrive/chadtech/spg.csvath, sequence\_length):

    # Load your solar power data from a CSV file or any other source

    # Assume the data has a 'timestamp' and 'power' column

    data = pd.read\_csv(data\_path, parse\_dates=['timestamp'], index\_col='timestamp')

    # Normalize the data using MinMaxScaler

    scaler = MinMaxScaler()

    data['power'] = scaler.fit\_transform(data['power'].values.reshape(-1, 1))

    # Create sequences for the RNN

    sequences, targets = create\_sequences(data['power'].values, sequence\_length)

    return sequences, targets, scaler

# Function to create input sequences and targets for the RNN

def create\_sequences(data, sequence\_length):

    sequences, targets = [], []

    for i in range(len(data) - sequence\_length):

        seq = data[i:i+sequence\_length]

        label = data[i+sequence\_length]

        sequences.append(seq)

        targets.append(label)

    return np.array(sequences), np.array(targets)

# Function to build the RNN model

def build\_rnn\_model(input\_shape, units=50):

    model = Sequential()

    model.add(LSTM(units, input\_shape=(input\_shape, 1)))

    model.add(Dense(1))

    model.compile(optimizer='adam', loss='mean\_squared\_error')

    return model

# Function to train the RNN model

def train\_rnn\_model(model, X\_train, y\_train, epochs=50, batch\_size=32):

    early\_stopping = EarlyStopping(monitor='loss', patience=5, restore\_best\_weights=True)

    model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, callbacks=[early\_stopping])

    return model

# Function to make predictions using the trained model

def make\_predictions(model, X\_test, scaler):

    predictions = model.predict(X\_test)

    predictions = scaler.inverse\_transform(predictions)

    return predictions

# Function to evaluate the model performance

def evaluate\_model(predictions, y\_test, scaler):

    y\_test = scaler.inverse\_transform(y\_test)

    mse = mean\_squared\_error(y\_test, predictions)

    rmse = np.sqrt(mse)

    return rmse

# Main function to execute the entire pipeline

def main():

    # Set your data path and sequence length

    data\_path = 'your\_data.csv'

    sequence\_length = 10

    # Load and preprocess data

    sequences, targets, scaler = load\_and\_preprocess\_data(data\_path, sequence\_length)

    # Split the data into training and testing sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(sequences, targets, test\_size=0.2, random\_state=42)

    # Reshape the data for compatibility with LSTM

    X\_train = X\_train.reshape((X\_train.shape[0], X\_train.shape[1], 1))

    X\_test = X\_test.reshape((X\_test.shape[0], X\_test.shape[1], 1))

    # Build and train the RNN model

    input\_shape = X\_train.shape[1]

    model = build\_rnn\_model(input\_shape)

    trained\_model = train\_rnn\_model(model, X\_train, y\_train)

    # Make predictions on the test set

    predictions = make\_predictions(trained\_model, X\_test, scaler)

    # Evaluate the model performance

    rmse = evaluate\_model(predictions, y\_test, scaler)

    print(f'Root Mean Squared Error (RMSE): {rmse}')

    # Plot the predictions against the true values

    plt.plot(y\_test, label='True Values')

    plt.plot(predictions, label='Predictions')

    plt.legend()

    plt.show()